

# R workflow for ABA data preparation

Jean-Matthieu Monnet

2021-02-05

The code below presents a workflow to prepare inventory data for the calibration of area-based models with airborne laser scanning data and field measurements.

Licence: CC-BY / Source page

Required R libraries : `ggplot2`, `sf`, `ggmap`

## Import field inventory data

### Tree-level inventory

96 plots of 15 m radius have been inventoried in the Quatre Montagnes area (Vercors Mountain, France). Plots are aggregated in clusters of 4. Data are provided in the folder “aba.model/field”:

- one file per plot for tree information (“Verc-CLUSTERID-PLOTID-ArbresTerrain.csv”)
- one file per cluster for plot center location (“Verc-CLUSTERID-PiquetsTerrain.csv”)

The first step is to import tree and plot information in two separate data.frames.

```
# set inventory parameters plot radius (m)
p.radius <- 15
# DBH threshold (cm) for inventory, trees smaller than this value are not
# inventoried
dbh.min <- 7.5
# list tree files by pattern matching
files.t <- as.list(dir(path = "./data/aba.model/field/", pattern = "Verc-[[[:alnum:]]{2}-[[[:digit:]]{3}]",
  full.names = TRUE))
# load content of all files with lapply
trees <- lapply(files.t, function(x) {
  # read table
  dummy <- read.table(x, sep = ";", header = T, stringsAsFactors = F)
  # add one column with plotId from file name
  cbind(dummy, data.frame(plotId = rep(substr(x, 30, 33), nrow(dummy))))
})
# bind elements of list into a single data.frame
trees <- do.call(rbind, trees)
# add study area info in plotId
trees$plotId <- paste0("Verc-", trees$plotId)
```

Fields are:

- `treeId`: tree id in the plot
- `poleId`: plot id inside the cluster (1 to 4)
- `Species`: tree species abbreviated as GESP (GENus SPecies)
- `Azimuth.gr`: azimuth in grades from the plot center to tree center
- `Slope.gr`: slope in grades from the plot center to tree center
- `Diameter.cm`: tree diameter at breast height (1.3 m, upslope), in cm
- `Ground.Distance.m`: ground distance between the plot center and tree edge at breast height, in m
- `Height.m`: tree height, in m
- `Appearance`:

- 0: lying or missing,
- 1: live,
- 2: live with broken treetop,
- 3: dead with branches,
- 4: dead without branches (snag)
- Tilted: 0: no, 1:yes
- Remark
- plotId: plot id

The following lines set column names to English and convert the `Appearance` column to factor with adequate values for the levels.

```
# change column names to English
names(trees) <- c("treeId", "poleId", "Species", "Azimuth.gr", "Slope.gr", "Diameter.cm",
  "Ground.Distance.m", "Height.m", "Appearance", "Tilted", "Remark", "plotId")
# add factor levels to column Appearance
trees$Appearance <- factor(trees$Appearance, levels = 0:4)
levels(trees$Appearance) <- c("missing or lying", "live", "live with broken treetop",
  "dead with branches", "dead without branches (snag)")
# convert species column to factor
trees$Species <- factor(trees$Species)
head(trees, n = 3)
```

```
##   treeId poleId Species Azimuth.gr Slope.gr Diameter.cm Ground.Distance.m
## 1      1      p1  PIAB         10        0    54.20817          6.00
## 2      2      p1  PIAB         12       -4    32.40395         12.13
## 3      3      p1  PIAB         28       -8    35.01409         13.90
##   Height.m Appearance Tilted Remark   plotId
## 1     33.8        live      0  170.3 Verc-01-1
## 2     26.8        live      0  101.8 Verc-01-1
## 3     29.1        live      0   110 Verc-01-1
```

## Plot-level information

The next step is to import plot coordinates.

```
# list location files by pattern matching
files.p <- dir(path = "./data/aba.model/field/", pattern = "Verc-[[:alnum:]]{2}_PiquetsTerrain.csv",
  full.names = TRUE)
# initialize data.frame
plots <- NULL
# load all plot files with a loop
for (i in files.p) {
  # read file
  dummy <- read.table(i, sep = ";", header = T, stringsAsFactors = F)
  # append to data.frame and add plotId info
  plots <- rbind(plots, cbind(dummy, data.frame(plotId = rep(substr(i, 30, 31),
    nrow(dummy)))))
}
# keep only necessary data in plots data.frame (remove duplicate position
# measurements)
plots <- plots[is.element(plots$Id, c("p1", "p2", "p3", "p4")), ]
# add plotId to clusterId
plots$clusterId <- paste0("Verc-", substr(plots$plotId, 1, 2))
plots$plotId <- paste(plots$clusterId, substr(plots$Id, 2, 2), sep = "-")
# keep only coordinates and Id in data.frame
plots <- plots[, c("X", "Y", "plotId", "clusterId")]
# convert to spatial sf object
plots.sf <- sf::st_as_sf(plots, coords = c("X", "Y"))
# set coordinate system
```

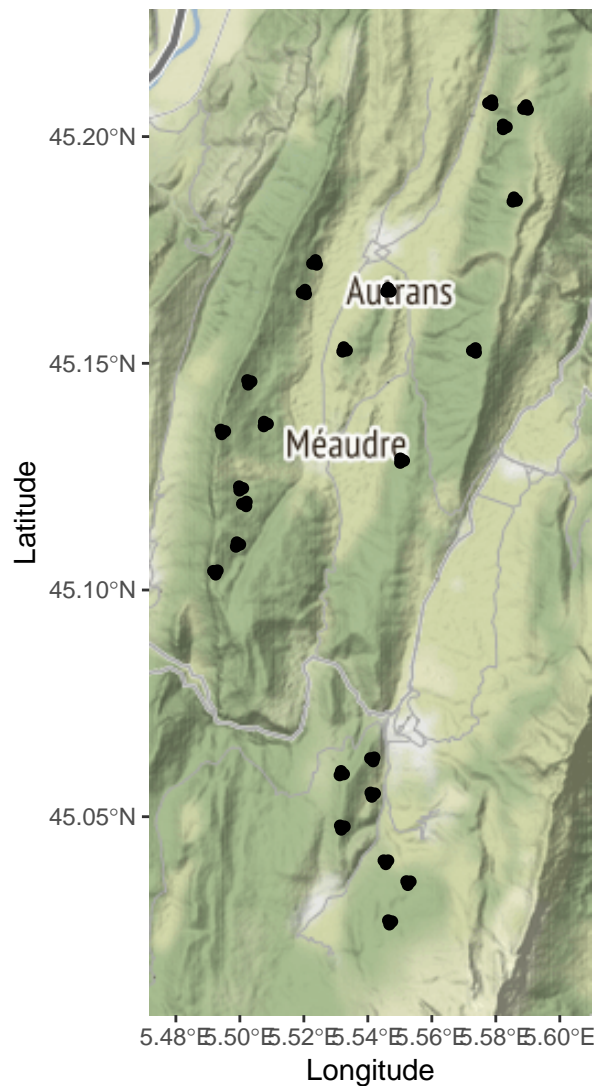
```
sf::st_crs(plots.sf) <- 2154
# add coordinates in data.frame of sf object
plots.sf <- cbind(plots.sf, data.frame(sf::st_coordinates(plots.sf)))
```

The following lines maps the plot locations with a background from OpenStreetMap :

- the bounding box of the region of interest is extracted in latitude / longitude values;
- the corresponding background is obtained with the package `ggmap`;
- `ggplot2` functions are used to plot the coordinates with this background.

```
# transform coordinates to lat / lon
plots.sf.transform <- sf::st_transform(plots.sf, 4326)
# extract bounding box
ext <- sf::st_bbox(plots.sf.transform)
# set buffer
buffer <- 0.02
# define location extent
loc <- c(ext$ymin - buffer, ext$xmin - buffer, ext$ymax + buffer, ext$xmax + buffer)
# add required names for location extent
names(loc) <- c("bottom", "left", "top", "right")
# load maps around plots
map <- ggmap::get_map(location = loc, source = "osm", zoom = 11)
# draw map
ggmap::ggmap(map) + ggplot2::labs(title = "Field plot location", x = "Longitude",
  y = "Latitude") + ggplot2::geom_sf(data = plots.sf.transform, inherit.aes = FALSE)
```

## Field plot location



In case tree positions have to be precisely calculated in a given projected coordinates system, it is necessary to correct magnetic azimuth with magnetic declination at the time of inventory and meridian convergence at the location. The values of magnetic declination and meridian convergence at each plot location are loaded from an additional table.

```
# get meta data about meridian convergence and declination by importing the
# metadata file.
meta <- read.table(file = "../data/aba.model/field/plot.metadata.csv", sep = ";",
  header = T)
# keep only required attributes
meta <- meta[, c("Id", "Convergence_gr", "Declinaison_gr")]
# rename attributes
names(meta) <- c("clusterId", "Convergence.gr", "Declination.gr")
# merge to add convergence and declination in plots data.frame
plots.sf <- merge(plots.sf, meta)
head(plots.sf, n = 3)

## Simple feature collection with 3 features and 6 fields
## geometry type: POINT
## dimension: XY
## bbox: xmin: 898137 ymin: 6455667 xmax: 898248.8 ymax: 6455708
## projected CRS: RGF93 / Lambert-93
```

```
##   clusterId   plotId      X      Y Convergence.gr Declination.gr
## 1   Verc-01 Verc-01-1 898205.7 6455708      -1.49862      0.83
## 2   Verc-01 Verc-01-2 898248.8 6455667      -1.49862      0.83
## 3   Verc-01 Verc-01-3 898137.0 6455687      -1.49862      0.83
##
##           geometry
## 1 POINT (898205.7 6455708)
## 2 POINT (898248.8 6455667)
## 3   POINT (898137 6455687)
```

## Compute tree coordinates

Tree coordinates can be computed from the plot center coordinates and from the azimuth, slope and ground distance measurements. In case ground distance is measured to the tree edge, the tree diameter has to be taken into account to compute the position of the tree center.

```
# merge tree and plots data.frames to import plot center
trees <- merge(trees, plots.sf[, c("X", "Y", "Convergence.gr", "Declination.gr",
  "plotId")], by = "plotId")
# compute projected coordinates
dummy <- lidaRtRee::polar2Projected(trees$X, trees$Y, 0, trees$Azimuth.gr/200 * pi,
  trees$Ground.Distance.m, trees$Slope.gr/200 * pi, trees$Declination.gr/200 *
  pi, trees$Convergence.gr/200 * pi, trees$Diameter.cm/100)
# add coordinates to trees data.frame
trees[, c("X", "Y", "Horiz.Distance.m")] <- dummy[, c("x", "y", "d")]
head(trees, n = 3)
```

```
##   plotId treeId poleId Species Azimuth.gr Slope.gr Diameter.cm
## 1 Verc-01-1     1    p1   PIAB      10      0     54.20817
## 2 Verc-01-1     2    p1   PIAB      12     -4     32.40395
## 3 Verc-01-1     3    p1   PIAB      28     -8     35.01409
##   Ground.Distance.m Height.m Appearance Tilted Remark      X      Y
## 1           6.00     33.8      live     0  170.3 898206.6 6455714
## 2          12.13     26.8      live     0  101.8 898207.9 6455720
## 3          13.90     29.1      live     0   110 898211.5 6455721
##   Convergence.gr Declination.gr           geometry Horiz.Distance.m
## 1      -1.49862      0.83 POINT (898205.7 6455708)      6.271041
## 2      -1.49862      0.83 POINT (898205.7 6455708)     12.268084
## 3      -1.49862      0.83 POINT (898205.7 6455708)     13.965465
```

## Import ALS data

The LAZ files provided with this tutorial are the point clouds already extracted around the plots. The code presented afterwards will thus extract the point clouds in files which correspond to already extracted data, but it can also perform the operation if the folder contains LAZ files corresponding to a wall to wall cover. LAZ files should be non-overlapping, and normalized (i.e. the Z field contains the height above ground).

## Point cloud extraction

Normalized points clouds are extracted over each plot. For the delineation of single trees, a buffer has to be added around the plot border. The attribute `buffer` is added and filled with `TRUE` for loaded points which are located outside of the plot extent. Those points will be removed when statistics are computed for the point cloud located inside the extent of the field plot.

```
# folder with laz files lazdir <-
# '/media/reseau/lessem/ProjetsCommuns/Lidar/data/38_Quatre_Montagnes/norm.laz/'
lazdir <- "../data/aba.model/ALS/plots.norm.laz/"
# make catalog of las files
cata <- lidR::catalog(lazdir)
# set coordinate system
```

```

sp::proj4string(cata) <- sp::CRS(SRS_string = "EPSG:2154")
# disable display of catalog processing
lidR::opt_progress(cata) <- FALSE
# extract from LAZ files in catalog
llas.height <- lidR::clip_circle(cata, plots.sf$X, plots.sf$Y, p.radius + 5)
# add 'buffer' attribute equal to TRUE to points outside plot
for (i in 1:length(llas.height)) {
  llas.height[[i]] <- lidR::add_attribute(llas.height[[i]], (llas.height[[i]]$X -
    plots.sf$X[i])^2 + (llas.height[[i]]$Y - plots.sf$Y[i])^2 > p.radius^2, "buffer")
}
# set names of point clouds in list
names(llas.height) <- plots.sf$plotId
# set negative height values to 0
for (i in 1:length(llas.height)) {
  llas.height[[i]]@data$Z[llas.height[[i]]@data$Z < 0] <- 0
}
# write plot clouds for (i in names(llas.height))
# {lidR::writeLAS(llas.height[[i]], file=paste0(lazdir, i, '.laz'))}

```

Points clouds with altitude values can be used to compute terrain statistics. The folder `plots.laz` contains the point clouds with altitude value extracted on the extent of each field plot. No buffer is added when loading those point clouds. The points not classified as ground are removed.

```

# folder with laz files
lazdir <- "../data/aba.model/ALS/plots.laz/"
# make catalog of las files
cata <- lidR::catalog(lazdir)
# set coordinate system
sp::proj4string(cata) <- sp::CRS(SRS_string = "EPSG:2154")
# disable plot of tile processing
cata@processing_options$progress <- FALSE
# extract las point cloud without buffer
llas.ground <- lidR::clip_circle(cata, plots$X, plots$Y, p.radius)
names(llas.ground) <- plots$plotId
# keep only ground points in point clouds
llas.ground <- lapply(llas.ground, lidR::filter_ground)

```

## Computation of terrain statistics

Terrain statistics can be extracted from the cloud of ground points with altitude values. The function `points2terrainStats` computes the aspect, altitude and slope of a point cloud by fitting a plane to points. The altitude is extracted by interpolating values at user-specified coordinates, or as the mean of the range of altitude values if no coordinates are specified.

```

# use mapply to apply points2terrainStats function to each point cloud while
# providing the coordinates of each center
terrain.stats <- mapply(function(x, y) {
  lidaRtRee::points2terrainStats(x, plots[y, c("X", "Y")], p.radius)
}, x = llas.ground, y = as.list(1:length(llas.ground)), SIMPLIFY = FALSE)
# bind results in data.frame
terrain.stats <- do.call(rbind, terrain.stats)
# compute terrain stats without specifying centre
terrain.stats2 <- lapply(llas.ground, lidaRtRee::points2terrainStats)
terrain.stats2 <- do.call(rbind, terrain.stats2)
# display results for comparison
head(cbind(terrain.stats, terrain.stats2), n = 3)

```

```

##           altitude azimuth.gr slope.gr adjR2.plane altitude azimuth.gr slope.gr
## Verc-01-1  1096.8      99.1    20.1         99.6   1096.7      99.1    20.1

```

```
## Verc-01-2    1083.4      94.0      16.1      99.2    1083.2      94.0      16.1
## Verc-01-3    1120.7     104.0     22.9      99.6    1120.4     104.0     22.9
##           adjR2.plane
## Verc-01-1           99.6
## Verc-01-2           99.2
## Verc-01-3           99.6
```

## Check field inventory data

Below are some checks that can be performed to correct possible errors in the field data.

### Consistency with the inventory protocol (all trees)

To ensure that all trees were correctly inventoried according to the protocol :

- If the distance of the tree to the plot center was recorded, the horizontal distance between the plot center and the tree center has to be checked to make sure the tree is included in the 15 meter radius.
- Diameters can also be checked to avoid that trees below the DBH limit remain in the inventory.

```
summary(trees[, c("Diameter.cm", "Horiz.Distance.m")])
```

```
##   Diameter.cm   Horiz.Distance.m
##   Min.      : 7.417   Min.      : 0.790
##   1st Qu.:11.332   1st Qu.: 7.241
##   Median :17.500   Median :10.476
##   Mean    :21.385   Mean    : 9.884
##   3rd Qu.:27.900   3rd Qu.:12.945
##   Max.    :95.100   Max.    :18.201
```

Once values have been checked for potential writing errors, remaining errors should be removed.

```
nb.trees <- nrow(trees)
# keep only trees inside plot
trees <- trees[trees$Horiz.Distance.m <= p.radius, ]
# keep trees above the DBH limit
trees <- trees[trees$Diameter.cm >= dbh.min, ]
# number of removed trees nb.trees - nrow(trees)
```

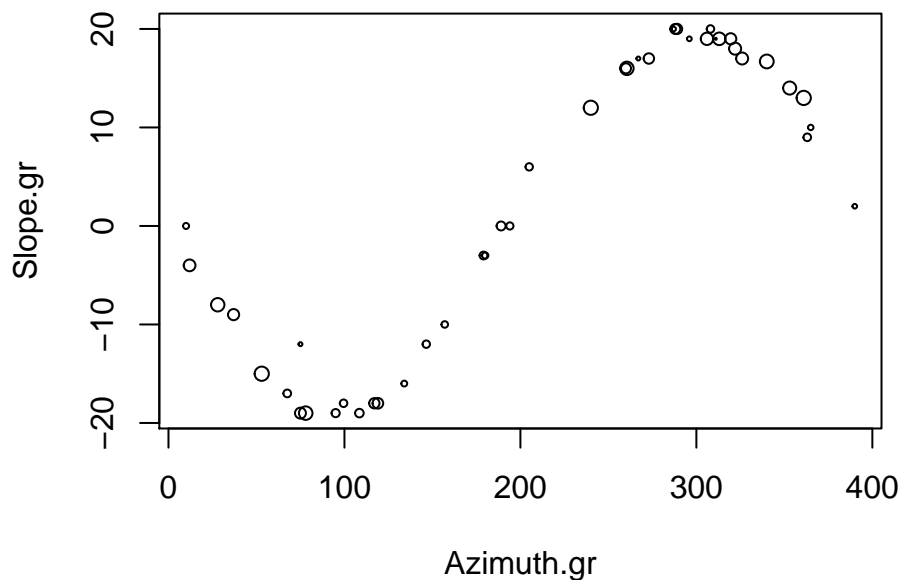
In this case 59 trees were removed.

### Consistency of trees azimuth and slope to center (by plot)

If tree 3D spherical coordinates have been recorded, azimuth can be checked against slope. For each plot, if the slope and azimuth are constant on the whole surface, then plotting the tree slope from the plot center against the tree azimuth from the plot center should draw a sinusoid-shaped point cloud. Outliers are likely to be errors in slope or azimuth values. For trees located close to the plot center, slope measurement precision is lower, so that larger deviation might be accepted.

```
# example plot to test
plot.test <- "Verc-01-1"
# plot slope as a function of azimuth, symbol size proportional to horizontal
# distance from center
plot(Slope.gr ~ Azimuth.gr, data = trees, subset = which(trees$plotId == plot.test),
     cex = Horiz.Distance.m/p.radius, main = "Trees slope and azimuth to plot center")
```

## Trees slope and azimuth to plot center



The following lines create a spatial polygon corresponding to one plot extent.

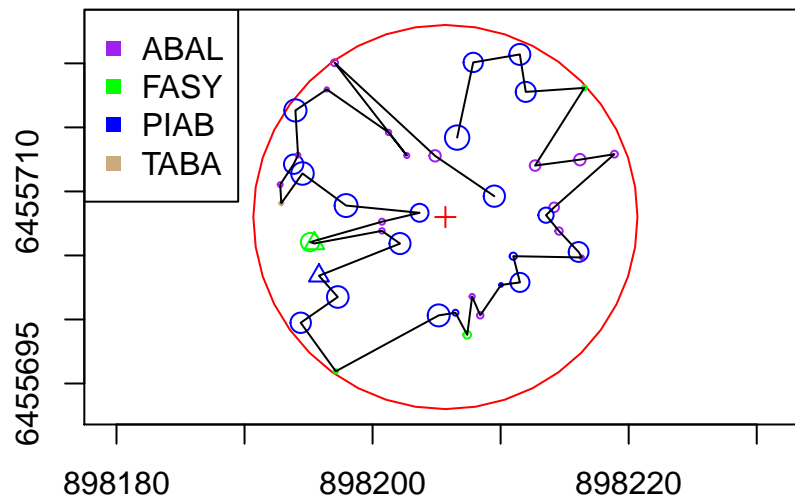
```
# extract example plot
plots.sf.t <- plots.sf[plots.sf$plotId == plot.test, ]
# create polygon with buffer of plot radius
plots.extent.t <- sf::st_buffer(plots.sf.t, p.radius, nQuadSegs = 10)
```

## Consistency of field operator path (by plot)

In case trees have been numbered in the order of inventory on the field, plotting tree positions with lines linking the trees in the order of inventory should trace the path of field operators. Any detour might indicate an error in distance or azimuth values.

```
# extraction of trees of plot to test
trees.t <- trees[trees$plotId == plot.test, ]
# display plot limits
plot(sf::st_geometry(plots.extent.t), border = "red", axes = TRUE)
# display plot center
plot(sf::st_geometry(plots.sf.t), col = "red", pch = 3, add = TRUE)
# add inventoried trees
lidaRtRee::plotTreeInventory(trees.t[, c("X", "Y")], trees.t$Height.m, species = as.character(trees.
  pch = as.numeric(trees.t$Appearance) - 1, add = TRUE)
# draw lines between trees
lines(trees.t[order(trees.t$treeId), c("X", "Y")])
```



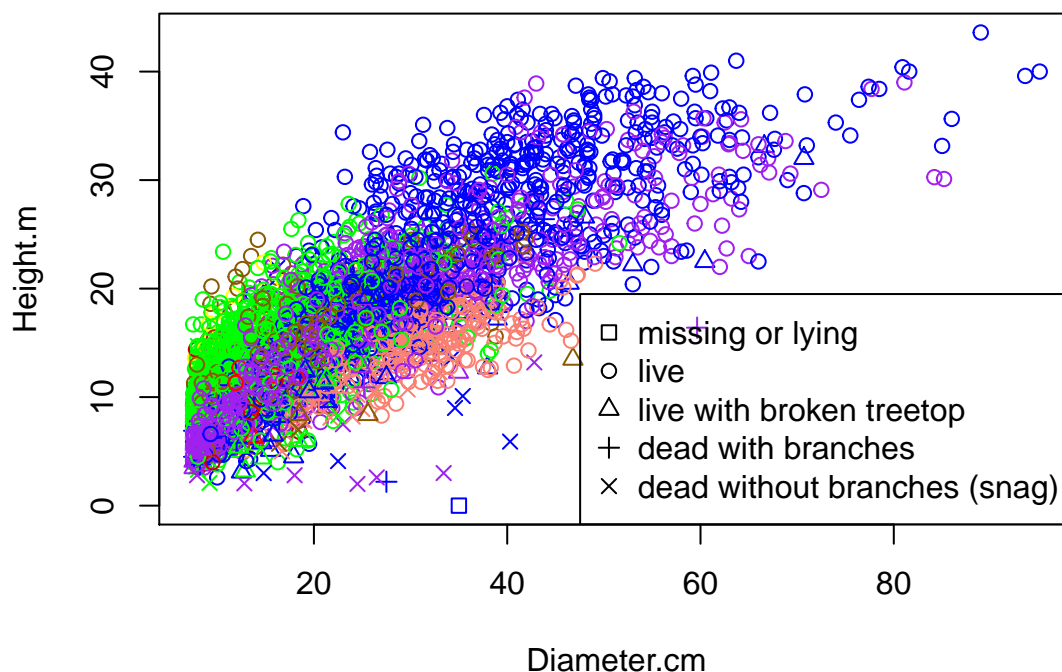


### Height / diameter allometry (all trees)

The allometry of trees can also be checked in case heights were also measured. Plotting the heights against diameters can be informative to detect errors in diameter or height values. In this case, the information about the appearance of trees sometimes provides explanation for outliers (mostly damaged or dead trees).

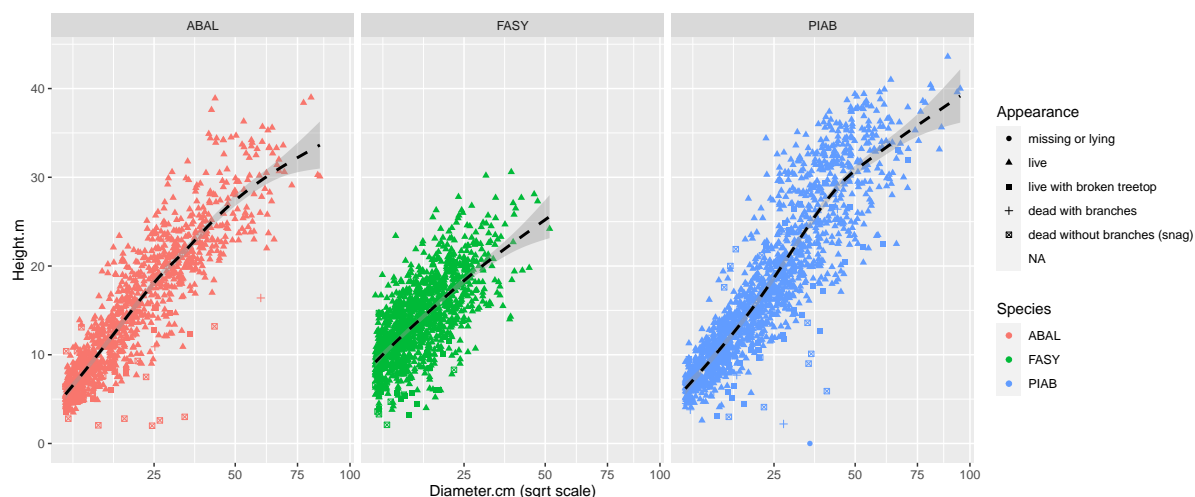
```
# symbol size proportional to horizontal distance from center
plot(Height.m ~ Diameter.cm, data = trees, col = lidaRtRee::speciesColor()[as.character(trees$Species)
  "col"], pch = as.numeric(trees$Appearance) - 1, main = "Height VS diameter, all trees")
legend("bottomright", legend = levels(trees$Appearance), pch = (1:length(levels(trees$Appearance)) -
  1))
```

## Height VS diameter, all trees



The `ggplot2` package provides useful syntax and functions for producing informative graphics. The next plot displays the diameter / height relationship for the three most abundant species in the dataset.

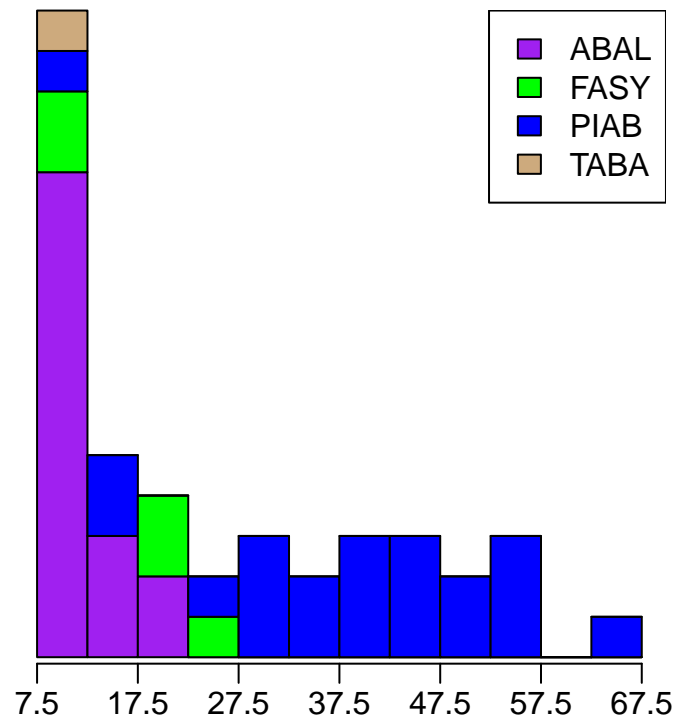
```
library(ggplot2)
g <- ggplot(trees[is.element(trees$Species, c("ABAL", "PIAB", "FASY")), ], aes(x = Diameter.cm,
  y = Height.m, shape = Appearance))
g + geom_point(aes(color = Species)) + scale_x_sqrt() + geom_smooth(aes(group = Species),
  color = "black", linetype = "dashed") + facet_grid(. ~ Species) + labs(x = "Diameter.cm (sqrt scale)")
```



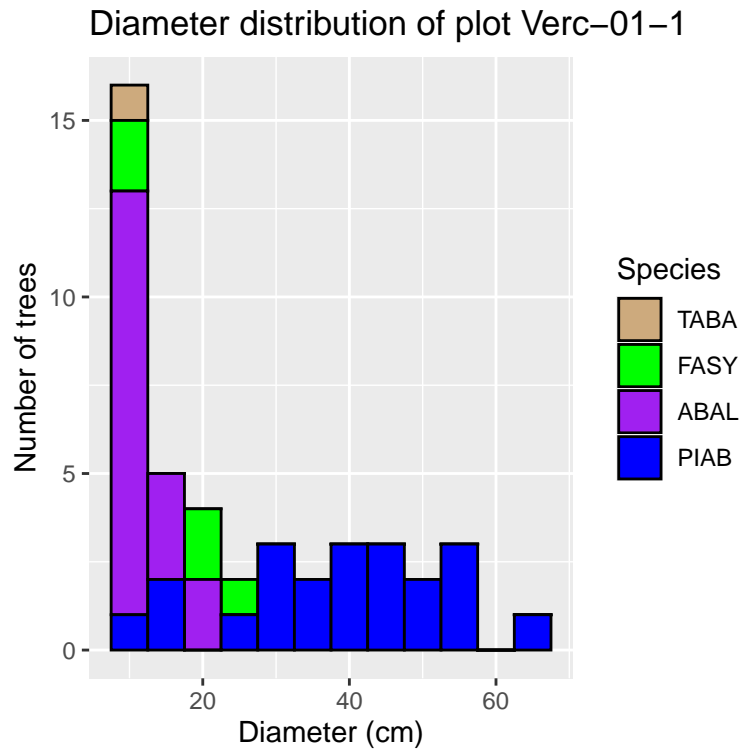
## Diameter distribution (by plot)

For each plot, displaying the histogram of trees, colored by species is also informative to make sure the forest structure is coherent.

```
# plot diameter distribution remove unused levels in species factor column
trees.t$Species <- factor(trees.t$Species)
# stacked histogram : data.frame split by species
dummy <- split(trees.t$Diameter.cm, trees.t$Species)
lidaRtRee::histStack(dummy, col = lidaRtRee::speciesColor()[names(dummy), "col"],
  breaks = seq(from = dbh.min, by = 5, to = 5 * ceiling(max(trees.t$Diameter.cm -
    2.5)/5) + 2.5), main = paste0("Diameter distribution of plot ", plot.test))
legend("topright", names(dummy), fill = lidaRtRee::speciesColor()[names(dummy), "col"])
```



```
# plot diameter distribution change order of levels to display more abundant
# species at the bottom
trees.t$Species <- factor(trees.t$Species, levels = names(sort(table(trees.t$Species))))
# load custom species table with associated colors
table.species <- lidaRtRee::speciesColor()
# extract species present on the plot
table.species <- table.species[levels(trees.t$Species), "col"]
# stacked histogram : data.frame split by species
ggplot(trees.t, aes(x = Diameter.cm, fill = Species)) + geom_histogram(breaks = seq(from = 7.5,
  to = max(trees.t$Diameter.cm) + 5, by = 5), color = "black") + scale_fill_manual(values = table.
  labs(title = paste0("Diameter distribution of plot ", plot.test), y = "Number of trees",
    x = "Diameter (cm)")
```



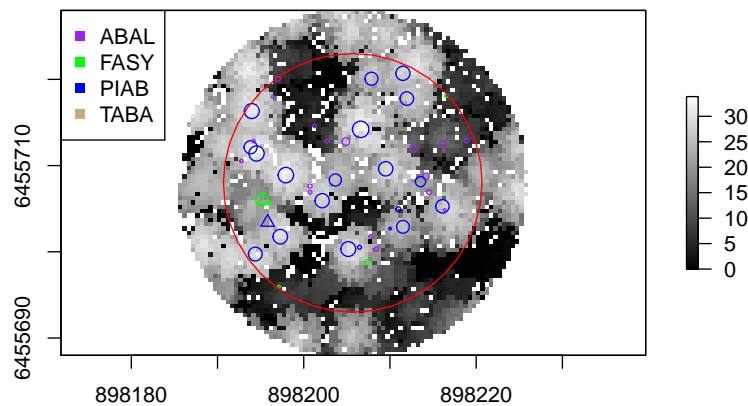
### Trees positions and remote sensing data (by plot)

In case remote sensing data is available, plotting trees positions, sizes and species on high-resolution remote sensing background also helps in identifying errors or incoherence. Airborne laser scanning data is particularly helpful. Please refer to the field plot coregistration and tree segmentation tutorials for additional information.

The tree positions can be plotted with the canopy height model computed with the ALS data.

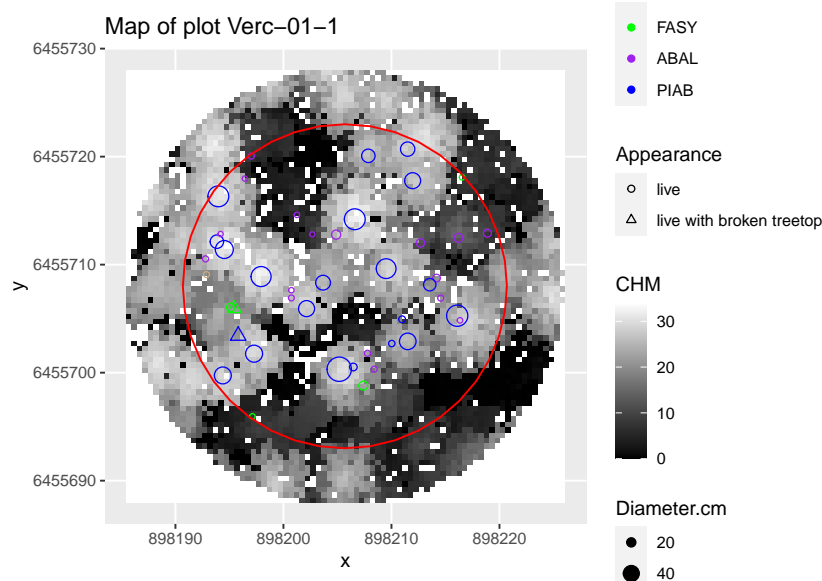
```
# compute background compute canopy height model
chm <- lidaRtRee::points2DSM(las.height[[plot.test]], res = 0.5)
# set coordinate system
raster::crs(chm) <- 2154
# display CHM
raster::plot(chm, col = gray(seq(0, 1, 1/255)), main = "Canopy Height Model and tree positions")
# add inventoried trees
lidaRtRee::plotTreeInventory(trees.t[, c("X", "Y")], trees.t$Height.m, species = as.character(trees.t$Species),
  pch = as.numeric(trees.t$Appearance) - 1, add = TRUE)
# display plot limits
plot(plots.extent.t, border = "red", color = NA, add = TRUE)
```

### Canopy Height Model and tree positions



Similar output with ggplot2.

```
library(ggplot2)
# convert raster to data.frame for display with ggplot
chm.df <- raster::as.data.frame(chm, xy = TRUE)
# rename columns
names(chm.df) <- c("x", "y", "CHM")
# create graph
g1 <- ggplot() + # raster background
  geom_raster(data = chm.df, aes(x = x, y = y, fill = CHM)) + # background scale in black and white
  scale_fill_gradientn(colours = c("black", "white"), na.value = "white") + # add trees
  geom_point(data = trees.t, aes(x = X, y = Y, shape = Appearance, size = Diameter.cm,
    color = Species)) + # shape radius (not area) proportional to diameter
  scale_radius(name = "Diameter.cm") + # custom colors for species
  scale_colour_manual(values = table.species) + # shapes are empty
  scale_shape(solid = FALSE) + # add plot extent
  geom_sf(data = plots.extent.t, fill = NA, color = "red") + # specify coordinate system
  coord_sf(datum = 2154) + # add title
  labs(title = paste0("Map of plot ", plot.test))
print(g1)
```



## Stand level parameters

### Compute stand level forest parameters

Three stand level parameters are computed by aggregating the tree level information at the plot level:

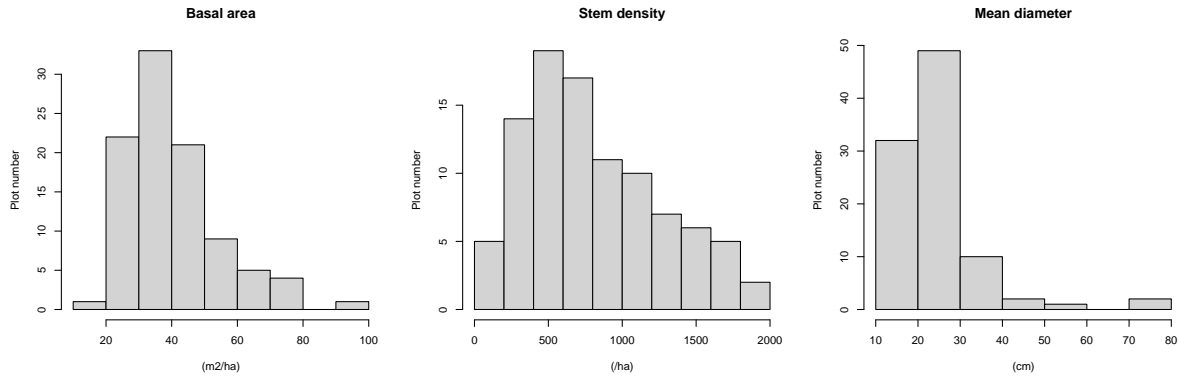
- basal area (G.m2.ha, m<sup>2</sup>/ha)
- mean diameter (D.mean.cm, cm)
- stem density (N.ha, /ha).

Only live trees are considered in this analysis.

```
# keep only live trees
trees <- trees[is.element(trees$Appearance, c("live", "live with broken treetop")),
]
# Basal area in m2/ha
dummy <- aggregate(Diameter.cm ~ plotId, trees, FUN = function(x) {
  sum(pi * (x/200)^2, na.rm = T) * (10000/(pi * p.radius^2))
})
# rename colum of result
names(dummy)[2] <- "G.m2.ha"
# add information to plots data.frame
plots <- merge(plots, dummy)
# Stem density in m2/ha
dummy <- aggregate(Diameter.cm ~ plotId, trees, FUN = function(x) {
  length(x) * (10000/(pi * p.radius^2))
})
names(dummy)[2] <- "N.ha"
plots <- merge(plots, dummy)
# Mean diameter in cm
dummy <- aggregate(Diameter.cm ~ plotId, trees, FUN = function(x) {
  mean(x)
})
names(dummy)[2] <- "D.mean.cm"
plots <- merge(plots, dummy)
# Summary statistics
summary(plots[, c("G.m2.ha", "N.ha", "D.mean.cm")])
```

##	G.m2.ha	N.ha	D.mean.cm
## Min.	:15.67	Min. : 56.59	Min. :14.70
## 1st Qu.:	:31.22	1st Qu.: 488.08	1st Qu.:19.55
## Median :	:37.35	Median : 714.43	Median :22.65
## Mean :	:40.17	Mean : 810.07	Mean :24.78
## 3rd Qu.:	:44.78	3rd Qu.:1110.55	3rd Qu.:26.31
## Max.	:99.18	Max. :1994.74	Max. :75.53

```
# display histograms
par(mfrow = c(1, 3))
hist(plots$G.m2.ha, main = "Basal area", xlab = "(m2/ha)", ylab = "Plot number")
hist(plots$N.ha, main = "Stem density", xlab = "(/ha)", ylab = "Plot number")
hist(plots$D.mean.cm, main = "Mean diameter", xlab = "(cm)", ylab = "Plot number")
```



### Add stratum information from external data

In this area, public forests are generally managed in a different way compared to private forests, resulting in different forest structures. Ownership is also linked to species composition. This classification could be used as a stratification when calibrating relationships between ALS metrics and forest parameters.

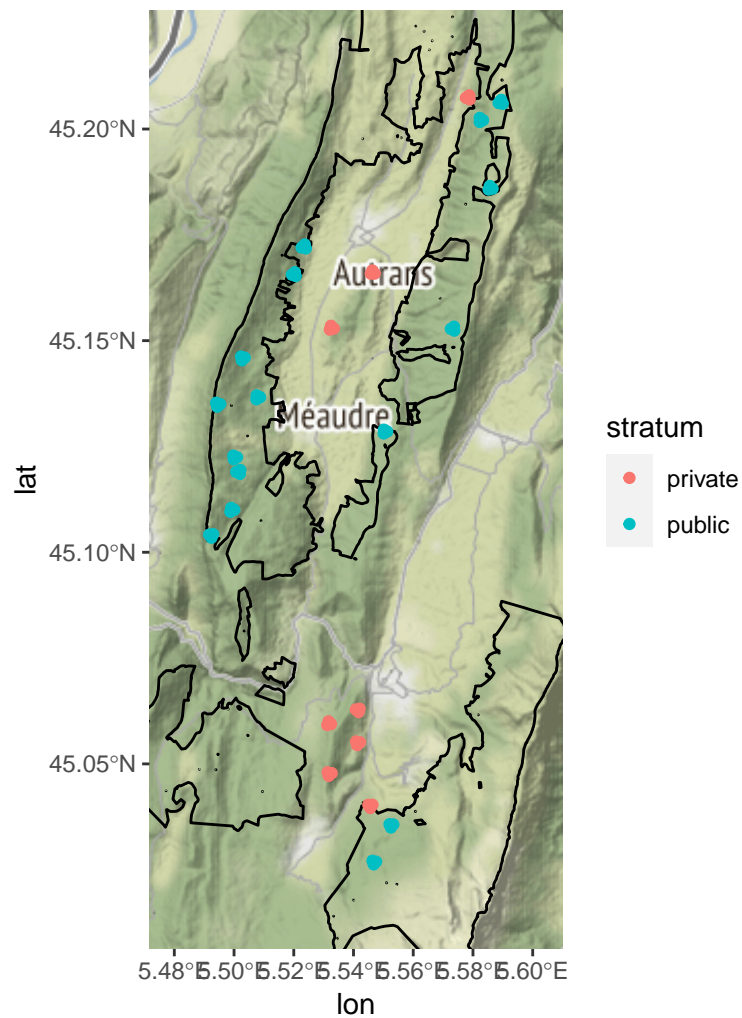
Plots will be attributed to strata based on external GIS layers. The first layer is a vector map of public forests (Forêts publiques, ONF Paris, 2019.) which is available under the Open Licence Version 2.0.

The following lines load the public forests layer and intersects it with the plot locations.

```
# load GIS layer of public forests
public <- sf::st_read("./data/aba.model/GIS/Public4Montagnes.shp", stringsAsFactors = TRUE,
  quiet = TRUE)
# set coordinate system
sf::st_crs(public) <- 2154
# spatial query
plots.sf <- sf::st_join(plots.sf, public)
# rename column and levels
plots.sf$stratum <- factor(ifelse(is.na(plots.sf$FID), "private", "public"))
# remove column
plots.sf$FID <- NULL
```

The following map displays the plots colored based on ownership, after spatial extraction of this information from the polygons of public forests (black borders).

```
# project points into map system
plots.sf.transform <- sf::st_transform(plots.sf, 4326)
public.transform <- sf::st_transform(public, 4326)
# display plots
ggmap::ggmap(map) + ggplot2::geom_sf(data = sf::st_geometry(public.transform), inherit.aes = FALSE,
  fill = NA, colour = "black") + ggplot2::geom_sf(data = plots.sf.transform, inherit.aes = FALSE,
  aes(color = stratum))
```



## Save data before next tutorial

The following lines save the data required for the area-based model calibration step.

```
# save(plots, file='./data/aba.model/output/plots.rda') save(llas.height,
# file='./data/aba.model/output/llas.height.rda', compress='bzip2')
# save(terrain.stats, file='./data/aba.model/output/terrain.stats.rda')
```